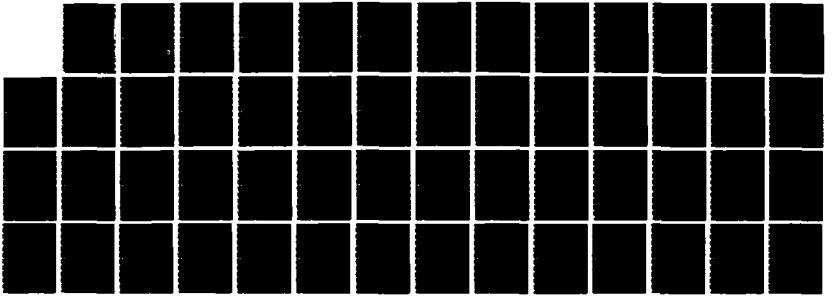
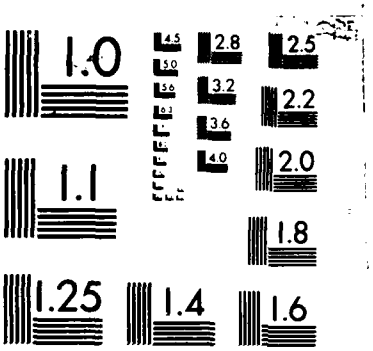
AD-A177 483 A I (ARTIFICIAL INTELLIGENCE) CENTER RESEARCH PROGRAMMER SUPPORT(U) JAYCOR VIENNA VA 15 SEP 86 N00014-85-C-2552 1/1

UNCLASSIFIED F/G 9/2 NL





(12)

A.I. Center
Research Programmer Support

Final Report

September 15, 1986

JAYCOR Contract #6255

In Response to:
Contract #N00014-85-C-2552

Prepared For:

Naval Research Laboratory
Washington, D.C. 20375-5000

Prepared By:

JAYCOR
1608 Spring Hill Road
Vienna, Virginia 22180

87 1 14 024

1. INTRODUCTION

JAYCOR is pleased to submit this final report summarizing the tasks performed by JAYCOR at the Navy Center for Applied Research in Artificial Intelligence (NCARAI) of the Naval Research Laboratory, under Contract #N00014-85-C-2552. This report gives an overview of the work performed to meet the tasks, location on NCARAI computers where the software is stored, documentation of the software, and pertinent information on any problems encountered during the performance of the tasks.

The report is organized on a task by task basis, each task briefly explained and the work summarized. Any software produced, obtained, or otherwise installed to satisfy task requirements is included in source form in Appendix. In addition to this summary final report, the reader is referred to the monthly reports submitted over the entire period of the contract to more thoroughly document the work performed.

2. TASKS AND WORK PERFORMED

In addition to the maintenance of the UNIX system on the VAX computers, JAYCOR implemented functions and programs for a number of peripherals. Background information on these peripherals and key points of interest about the software is summarized below. If the software was modified from previously existing software rather than completely

rewritten by JAYCOR personnel, pointers to the original source are also given. Location of online source code and documentation is noted.

2.1. 3/4" VIDEO TAPE PLAYER (VAX and LMI)

During the previous system programmer contract period, the AI Center took delivery on a programmable 3/4" video tape player. This player has much the same functionality as the 1/2" player which JAYCOR programmed under that contract. This software was thus usable in very slightly modified form with the larger tape size.

To use and program the player from the VAX, the user connects it to the TERABIT computer interface according to the TERABIT manual. To use from the LMI LAMBDA computer, the user connects the player to the TERABIT interface, then connects the TERABIT interface to the LAMBDA serial port labeled "SDU-Serial-B". One then runs the software written by JAYCOR personnel under the LISP environment as documented in the manual in the Appendix. Each function can be individually invoked or can be combined into a complex program of function calls as suits the user.

All software for this task is contained in the VAX-11/780 directory "/aic2/smith/contracts/sys2" under appropriate subdirectories. Additional copies of the LMI-specific software are also available on the LMI computer.

2.2. 1/2" VIDEO TAPE AND DISK PLAYERS (LMI)

The tape and disk player software programmed under the previous contract for the VAX was transported to the LMI computer both to allow continuity of interface and to prevent duplication of previous programming effort. During the transportation modifications to the FRANZ LISP code were made to make it COMMON LISP compatible. Full functionality was maintained during this transfer.

The user interface is quite similar for both of these peripherals to that of the 3/4" tape player, hence users are not presented with a confusing change in command structure. The Appendix contains copies of the manuals with small examples of usage demonstrating this similarity.

The software and documentation for these peripherals is located in the VAX-11/780 directory "/aic2/smith/contracts/sys2" under appropriate subdirectories. Where duplicate copies of software were possible, hard file links have been used instead.

2.3. RACAL-VADIC MODEM DRIVER (LMI)

The LMI LAMBDA computers come with a version of the public domain KERMIT program. This program uses whatever modem (or simple connection) is attached to the "SDU-Serial-B" rs232 port. For this task, JAYCOR personnel connected the Racal-Vadic modem to the serial port, enabled the port by running the KERMIT program, and tested out the

functionality of the modem through connections from the LMI to the VAX computers. The KERMIT program was successfully tested by transferring files from the LMI to the main VAX computer. All Racal-Vadic internal operations were accessible through the KERMIT software interface.

To use the KERMIT program while logged into the LMI computer, type in <SYSTEM>K, thus starting the KERMIT program. Then, using the mouse in the standard manner (explained in LMI documentation), "click" on the "Connect" menu selection. One can then use the Racal-Vadic modem programming commands as if connected through a terminal.

The KERMIT software (enabled, tested, and verified, but NOT supplied by JAYCOR) is located on the LAMBDA computer under the system source directory. Due to its large size it is not duplicated in this summary report.

2.4. LA50 PRINTER DRIVER (LMI)

Two versions of printer drivers were written by JAYCOR personnel for this task. Both versions are capable of printing rectangular screen areas onto an LA50 printer connected to the "SDU-Serial-B" rs232 port. The areas give a somewhat accurate graphic rendition of the actual screen display, however the aspect ratio is, as expected, not 1:1.

One major problem with the use of the serial port and the LA50 printer was uncovered during the implementation of this software, namely the lack of adequate flow control

during the printing process. For some larger sized rectangular regions, the LA50 can not keep up with the LAMBDA's stream of data, hence attempts to use XON/XOFF flow control to synchronize. Unfortunately, the LAMBDA's serial port software does not use flow control in its current configuration so some data can be lost, resulting in both skew of and superfluous patterns in the output image.

The utility of the LA50-based printing has been superceded by the much higher resolution printing available with the Ethernet-connected IMAGEN laser printer. This printer produces much finer images much more quickly over the high speed network link. All software is kept on the LAMBDA computer.

2.5. RESTRICTED SHELL

As JAYCOR noted in its proposal for the subject contract, this task is met by having a "shell" field in the password entry for a user which will only invoke a specific program, logging off on exit. Due to the nature of the library system installed by the NCARAI, this method of access restriction will not be necessary at this writing, instead a unique login ID and password are used with NRL's library system (explained in the following sections dealing with library tasks).

2.6. RAMTEK SOFTWARE

Under the initial system programmer contract awarded to

JAYCOR, graphics functions were implemented which allowed users to interface their C programs to the Ramtek devices on both the VAX-11/780 and VAX-11/750 (ATE machine). This software provided line and rectangular region graphics as well as text in colors selected by the user. These functions have been extended and an interface to the FRANZ LISP environment has been implemented to meet the requirements of this task. In addition, certain image processing functions have been implemented and a sample set of images obtained for testing purposes.

All software for this task is stored in the VAX-11/780 directory "/aic2/smith/contracts/sys2" under an appropriate subdirectory. The functions typically are all self documenting due to their simple usage, but other documentation is also in the same location.

2.7. LIBRARY DATABASE

The library database was tested thoroughly during the initial period of this contract. After this testing and communication with other libraries in the area (Library of Congress, NRL main library), it was decided to obtain the standardized OCLC/DIALOG/LS2000 accessing permissions, thus maintaining compatibility with libraries throughout the country. An OCLC terminal (actually an IBM PC with appropriate hardware and software enhancements) was received and installed. It has proven to be a valuable asset to the library. When the main NRL library LS2000 system comes

fully online, the NCARAI library will be able to access and use the main NRL database for such things as verifying the availability of references, checking out of references, and doing subject/author/keyword searches. Arrangements have been made with the main library to obtain "zebra codes" for the current references in the NCARAI library. These will be attached to the spines of the books and allow the computerized checking out of references to be performed.

2.8. LIBRARY ORGANIZATION

The library has undergone considerable rearrangement since contract award. References have been organized to more closely follow standard library practice, circulation for periodicals has been totally redone to provide more timely circulation, and the physical arrangement of the library has been changed to allow easier access to the stacks.

In addition to the above, the design of an expanded library has been submitted to the COTR. This new library takes into account the tremendous influx of new references and the expanding space necessary to hold them. Using the current "community" room in the AI Center, the library will be able to handle acquisitions at the current rate for a number of years. At the same time, the study atmosphere will be improved significantly through the presence of natural light from the large window area and the ability to use study carrels.

2.9. DOCUMENT ORGANIZATION

A large number of documents have been ordered and received from DTIC. These documents comprise the collection of works which in some manner reference AI. The documents were obtained in both hardcopy and Microfiche form. The current plan for cataloging these documents is to order the DTIC listing from which they were ordered. This would be in addition to the ordered storage of both the fiche and the hardcopy (see the Inventory section below).

2.10. POINT OF CONTACT

Throughout the period of the contract the JAYCOR librarian has interacted considerably with the user community. This interaction has covered the range of duties from simple reference lookup to the arrangement of interlibrary loans. Professional relationships have been developed with the NRL library as well as with the Library of Congress. These relationships are ongoing.

2.11. INVENTORY AND REQUIREMENTS ANALYSIS

The JAYCOR librarian has been closely involved with the COTR in performing a requirements analysis for the library. From this analysis the needs of the library were discovered to exceed current library capabilities. This led to the planning of the new expanded library space using the NCARAI community room. This room will be much more conducive to study and research.

Another product of the library analysis was the acquisition of a Microfiche hardcopy printer. This machine has been very well received. Using standard fiche, the user can view a document and, if desired, make a hardcopy version at will. Through the use of fiche for much of the documents received by the Center, considerable space will be saved.

APPENDIX

This Appendix contains copies of the source code of all software used to satisfy the requirements of the tasks of the Statement of Work. This software was either written, modified, or acquired by JAYCOR as noted.

VIDEO TAPE PLAYERS

Video Tape Player

Naveen Hota

JAYCOR

Usage

The video tape player can be operated from lisp. The routines are written in C. These routines should be loaded in to lisp before using them. To use them do the following:

(1) switch on the Terabit (VM-820).

(2) switch on the video player.

(3) if a SONY tape player is used, set remote select switch on the back side of the tapeplayer to 300 position.

On VAX

```
(load 'tapefns)
(vp-open)
--- whatever ---
(vp-close)
```

On LMI

```
(load 'tapefnslm)
(with-open-file (*video* "sdu:serial-b:")
     (send *video* ':set-baud-rate 9600)
--- whatever ---
)
```

Some times this won't work due to the serial-port being opened by someone and forgot to close. In that case use the following:

```
(with-open-file (*video* (steal-port-b)
     (send *video* ':set-baud-rate 9600)
     ---- whatever ---- )
```

On Symbolics

```
(load 'discfnslm)
(with-open-stream (*video* (si:make-serial-stream ':unit 2
':baud 9600 ':check-over-run-errors t ':ascii-characters t
':force-output t))
   ----- whatever -----)
```

Commands

Following is a description of each of the functions that can be called from lisp. All the commands return an integer code. A value of 0 indicates success. For other values, an error message is printed and an integer between 1 and 9 is returned.

Name: **vp-retid**

Arguments: none

Description: vp-retid is always called (internally) after the execution of each command. This function returns an integer indicating the status. A value of 0 indicates success. For any value other than 0, an appropriate error message is printed and an interger is returned.

Name: **vp-open**

Arguments: none

Description: Video player open (only on VAX).
Enables device communication.

Name: **vp-close**

Arguments: none

Description: Video player close (only on VAX).
Disables device communication.

Name: **vp-home**

Arguments: none

Description: Home.
To place VM-820 in a known state. This should be the first command sent to VM-820.

Name: **vp-initialize**

Arguments: none

Description: Initialize.
Positions the tape at the first valid frame.
Audio and video are turned off during this command.

Name: **vp-quit**

Arguments: none

Description: Quit
Terminates active command.

Name: **vp-play-forward**

Arguments: n1 n2

Description: Play forward.
Plays the tape from n1 to n2.
both n1 and n2 should be integers.
n2 = 0 => plays till the end.(default)

n1 < 0 => plays from the beginning frame.
n1 = 0 => plays from the current frame.(default)

Name: **vp-fast-forward**

Arguments: none

Description: Fast forward.
Fast forwards toward the end of the tape.

Name: **vp-rewind**

Arguments: none

Description: Rewind.
Rewinds toward the beginning of the tape.

Name: **vp-go-to-frame**

Arguments: n1 n2

Description: Go to a frame number.
Advances to the frame number n1.(default: beginning)
n2 specifies the video end condition.
n2 = 0 => video stops after reaching n1.(default)
n2 = 1 => video plays after reaching n1.

Name: **vp-single-frame**

Arguments: none

Description: Single frame.
Plays forward a single frame.

Name: **vp-frame-number**

Arguments: none

Description: Current frame number.
Returns the current frame number.

Name: **vp-hold-enable**

Arguments: none

Description: Hold enable.
Pauses the tape at the current location.

Name: **vp-hold-disable**

Arguments: none

Description: Hold disable.
Unpauses the tape.

Name: **vp-last-command**

Arguments: none

Description: Last command.
Returns the last active command.
0 - no active command

1 - vd-FF (fast forward)
2 - vd-G (Go to frame)
3 - vd-R (reject)
4 - vd-I (initialize)
5 - vd-PF (play forward)
6 - vd-HE (hold enable)
7 - vd-HE (hold disable)
8 - vd-SF (single frame)
9 - vd-FR (rewind).

Name: **vp-last-command-status**

Arguments: none

Description: Last command status.
Returns the status of the last command.
A value of 0 indicates completion.

Name: **vp-environment-read**

Arguments: none
(lispmc can take an optional arg: oneof (video audio1 audio2 videoend continuous)

Description: Environment read.
Returns the present environment setting as an integer.
Returned integer positions = 87654321.

position	description (1-on : 0-off)
8	video
7	audio ch.1
6	audio ch.2
5	video end condition.
4	Continuous Frames.
3,2,1	-- always 0s (not used).

on lispmc
if noarg, returns a list (videoststus ch1st' ch2sta' videoendst' contsta')
if arg returns arg's status as 0/1.

Name: **vp-getc**

Arguments: arg1

Description: Get the current status of arg1.
Returns the present environment setting of arg1 as t/nil indicating on/off.
Arg1 should be one of the following:
video, audio1, audio2, videoend, continuous.

ote:R only on VAX. For lispmc, see *vp-environment-read.*

Name: **vp-environment-write**

Arguments: n1 n2 n3 n4 n5

Description: Environment write.
To write the environment settings.
n1 - video (1-on, 0-off)
n2 - audio ch.1 (1-on, 0-off)
n3 - audio ch.2 (1-on, 0-off)
n4 - video end cond. (1-on, 0-off)
n5 - Continuous frame. (1-on, 0-off)

Name: **vp-setc**

Arguments: arg1 arg2

Description: Set the condition of arg1 to arg2.
Arg1 should be one of the following:
video, audio1, audio2, videoend, continuous.
Arg2 should be on/off/1/0.
Returns 0.

Name: **vp-mode-computer**

Arguments: none

Description: Mode computer.
To set the VM-820 to computer mode.
This is the initial mode after the power on of the VM-820 or a reset.

Name: **vp-mode-terminal**

Arguments: none

Description: Mode terminal.
To set the VM-820 to terminal mode.

Name: **vp-sendnum**

Arguments: n1

Description: Sends a number (integer) n1.
vp-sendnum is used to send an integer.

Name: **vp-eject-tape**

Arguments: none

Description: Eject tape.
Unloads the tape.

Name: **vp-putchars**

Arguments: n1

Description: send the chars obtained by evaluating n1 (VAX).
lispmc: n1 should be list and sends all the chars in n1.
Used to explicitly send chars corresponding to commands.

Example: (vp-putchars 'FF@) to fast forward the tape.

Note: Carriage return is denoted by @ character.

Name: **vp-getchars**

Arguments: n1

Description: Reads n1 chars from the video line.
n1 = 0 implies read till end of line.
Example: (vp-getchars 4) to read 4 chars from the video line.

Note:

```lisp
;       ====  tapefns.l  ====
;
;  These functions are used to load in the C code and bind LISP
;  function names to the particular C routines. Run once at startup time
;


(cfasl '/usr/local/lib/video/vplay.o '_v_open 'vp-open "integer-function")

(getaddress '_v_close 'vp-close "integer-function")
(getaddress '_SetTerm 'vp-set "integer-function")
(getaddress '_RstTerm 'vp-reset "integer-function")
(getaddress '_v_retid 'vp-retid "integer-function")
(getaddress '_v_sendnum 'vp-sendnum "integer-function")
(getaddress '_v_FF 'vp-fast-forward "integer-function")
(getaddress '_v_FR 'vp-rewind "integer-function")
(getaddress '_v_G 'vp-I-go-to-frame "integer-function")
(getaddress '_v_HD 'vp-hold-disable "integer-function")
(getaddress '_v_HE 'vp-hold-enable "integer-function")
(getaddress '_v_I 'vp-initialize "integer-function")
(getaddress '_v_PF 'vp-I-play-forward "integer-function")
(getaddress '_v_R 'vp-eject-tape "integer-function")
(getaddress '_v_SF 'vp-single-frame "integer-function")
(getaddress '_v_H 'vp-home "integer-function")
(getaddress '_v_ER 'vp-environment-read "integer-function")
(getaddress '_v_v_getc 'vp_v_getc "integer-function")
(getaddress '_v_EW 'vp-environment-write "integer-function")
(getaddress '_v_v_setc 'vp_v_setc "integer-function")
(getaddress '_v_L 'vp-last-command "integer-function")
(getaddress '_v_MC 'vp-mode-computer "integer-function")
(getaddress '_v_MT 'vp-mode-terminal "integer-function")
(getaddress '_v_N 'vp-frame-number "integer-function")
(getaddress '_v_Q 'vp-quit "integer-function")
(getaddress '_v_? 'vp-last-command-status "integer-function")
(getaddress '_v_ptchs 'vp_pchs "integer-function")
(getaddress '_v_gtchs 'vp_gchs "integer-function")
(getaddress '_v_flush 'vp-flush "integer-function")


(defun mapname(x1)
    (cond ((equal x1 'video) 1)
          ((equal x1 'audio1) 2)
          ((equal x1 'audio2) 3)
          ((equal x1 'videoend) 4)
          ((equal x1 'continuous) 5)
          (t (msg "Illegal first argument to vp-setc/getc ") 0)))

(defun vp-getc(x1)
    (= -1 (vp_v_getc (mapname x1))))

(defun vp-setc(x1 x2)
    (vp_v_setc (mapname x1) (on-off x2)))

(defun on-off(st)
    (cond ((equal st 'on) 1)
          (t 0)))
(defun vp-go-to-frame fexpr(l)
    (cond ((null l) (vp-I-go-to-frame 0 0))
          ((= (length l) 1) (vp-I-go-to-frame (car l) 0))
          (t (vp-I-go-to-frame (car l) (cadr l)))))
```

```lisp
(defun vp-play-forward fexpr(l)
    (cond ((null l) (vp-I-play-forward -1 0))
          ((= (length l) 1) (vp-I-go-to-frame (car l) 0))
          (t (vp-I-go-to-frame (car l) (cadr l)))))
; -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
(defun vp-putchars (cchars)
    (vp_pchs (length (exploden cchars))
         (apply 'vectori-byte (aexploden cchars))))


(defun vp-getchars (num)
    (let ((chars (new-vectori-byte 50 "^"))
          (ans)
          (tl 0))
         (vp_gchs num chars)

         (while (and (< tl 50) (not (= (vrefi-byte chars tl) 94)))
                (setq ans (cons (vrefi-byte chars tl) ans)
                      tl (+ tl 1)))
         (reverse (apply 'append
                         (@ '(lambda(x)
                                  (cond ((= x 94) nil)
                                        (t (list (maknam (list x))))))))))))


; -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

(defun vp-help()
    (msg (N 1) "*** The following functions are available ***")
    (msg (N 1) "vp-open                   " "vp-close             " "vp-retid ")
    (msg (N 1) "vp-home                   " "vp-initialize        " "vp-quit")
    (msg (N 1) "vp-play-forward           " "vp-fast-forward      " "vp-rewind")
    (msg (N 1) "vp-go-to-frame            " "vp-single-frame      " "vp-frame-number ")
    (msg (N 1) "vp-hold-disable           " "vp-hold-enable       " "vp-last-command ")
    (msg (N 1) "vp-last-command-status " "vp-environment-read " "vp_getc " )
    (msg (N 1) "vp-environment-write      " "vp-setc              " "vp-mode-computer ")
    (msg (N 1)  "vp-mode-terminal         " "vp-sendnum           " "vp-eject-tape")
    (msg (N 1)          "vp-putchars              " "vp-getchars"))
```

```
/*
        vplay.c

        Loaded in by the lisp calls in "tapefns.l", these are the actual
        routines that interface to tthe tape players. All LISP functions
        call these to do their work. Compile with:

                cc -O -c vplay.c

*/

#include <sgtty.h>
#define RETURN '\015'
static int video;
static char buff[25];

/* to open the video player */
v_open()
{
        int SetTerm();
        if((video = open("/dev/ttyi0", 2)) < 0) return (-1);
        SetTerm();
        return(0);
}

/* to close the video player */
v_close()
{
        int RstTerm();
        RstTerm();
        close(video);
        return(0);
}

struct sgttyb oldmodes,newmodes;

/* to set the line to rawmode */
SetTerm()
{
        ioctl(video, TIOCGETP, &oldmodes);
        ioctl(video, TIOCGETP, &newmodes);
        newmodes.sg_flags  = RAW;
        newmodes.sg_flags &= ~ECHO;
        newmodes.sg_ispeed = B9600;
        newmodes.sg_ospeed = B9600;
        ioctl(video, TIOCSETN, &newmodes);
}

/* to set the line back to normal mode */
RstTerm()
{
        ioctl(video, TIOCSETN, &oldmodes);
}

/* to get the return id and print the err msg, if any */
int v_retid()
{
        int c_to_i();
        read(video,buff,1);
```

```
        switch (buff[1]){
        case '*'
        case '0'
                return(0);
        case '1'
                printf("Illegal Command for this machine: %c %c %c\n",
                buff[0],buff[1],buff[2]);
                return(1);
        case '2'
                printf("Bad command format: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(2);
        case '3'
                printf("Bad option format: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(3);
        case '4'
                printf("Last active command is still active: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(4);
        case '5'
                printf("Command failed: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(5);
        case '8'
                printf("Bad machine code: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(8);
        case '9'
                buff[0] = buff[2];
                read(video,buff+1,8);
                return(c_to_i(buff,8));
        default
                printf("returning undefined code - %c\n",buff[1]);
                return(7);
        }
}

/* to change char to integer */
int c_to_i(s,l)
char s[];
int l;
{
        int i = 0;
        int n = 0;
        while (s[i] >= '0' && s[i] <= '9' && i < l){
                n = 10* n + s[i] - '0';
                i++;
        }
        return(n);
}


/* to send a number */
int v_sendnum(x)
int *x;
{
        int     bi = 16;
        int     y;
        char    tbuff[15];
```

```
        y = *x;
        while (y > 0 && bi > -1) {
                tbuff[bi--] = y % 10 + '0';
                y /= 10;
        }
        write(video, tbuff+bi+1, 14-bi);
}

/* fast forward */
int v_FF()
{
        int v_retid();
        buff[0] = 'F';
        buff[1] = 'F';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
        return(v_retid());
}

/* fast reverse */
int v_FR()
{
        int v_retid();
        buff[0] = 'F';
        buff[1] = 'R';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
        return(v_retid());
}

/* go to a frame number
   n1 is frame number, n2 is on off flag */
int v_G(n1, n2)
int *n1, *n2;
{
        int v_retid();
        buff[0] = 'G';
        buff[1] = '/';
        buff[2] = 'E';
        buff[3] = (*n2 > 0) ? '1' : '0'; /* play or stop */
        buff[4] = ',';
        buff[5] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,5);
        v_sendnum(n1);
        write(video, buff+5, 1);
        return(v_retid());
}

/* hold disable */
int v_HD()
{
        int v_retid();
        buff[0] = 'H';
        buff[1] = 'D';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
```

```
        return(v_retid());
}

/* hold enable */
int v_HE()
{
        int v_retid();
        buff[0] = 'H';
        buff[1] = 'E';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
        return(v_retid());
}


/* initialize */
int v_I()
{
        int v_retid();
        buff[0] = 'I';
        buff[1] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,2);
        return(v_retid());
}

/* play forward
   n1 is frame number to start, n2 is frame number to end
   n1 < 0 starts from begining, n2 = 0 stops at the end
   n1 = 0 starts from the current frame */
int v_PF(n1, n2)
int *n1, *n2;
{
        int v_retid();
        buff[0] = 'P';
        buff[1] = 'F';
        buff[2] = ':';
        buff[3] = ',';
        buff[4] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,2);
        if(*n1 == 0){
                write(video,buff+4,1);
                return(v_retid());
        }
        if(*n1 < 1) {*n1 = 0;}
        write(video,buff+2,1);
        v_sendnum(n1);
        if (*n2 > *n1){
                write(video, buff+3, 1);
                v_sendnum(n2);
        }
        write(video, buff+4, 1);
        return(v_retid());
}

/* reject */
int v_R()
{
```

```
        int v_retid();
        buff[0] = 'R';
        buff[1] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,2);
        return(v_retid());
}

/* single step forward a frame */
int v_SF()
{
        int v_retid();
        buff[0] = 'S';
        buff[1] = 'F';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
        return(v_retid());
}

/* home VM -820 */
int v_H()
{
        buff[0] = '@';
        buff[1] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,2);
        read(video,buff,1);
        if (*buff == '*') return(0);
        printf("returning undefined code : %c\n",*buff);
        return(7);
}

/* environment read */
int v_ER()
{
        int v_retid();
        buff[0] = 'E';
        buff[1] = 'R';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
        return(v_retid());
}

/* v_v_getc */
int v_v_getc(n1)
int *n1;
{
        buff[0] = 'E';
        buff[1] = 'R';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
        read(video,buff,3);
        switch (buff[1]){
        case '*' :
        case '0' :
                return(0);
        case '1' :
```

```
                printf("Illegal Command for this machine: %c %c %c\n",
                buff[0],buff[1],buff[2]);
                return(1);
        case '2' :
                printf("Bad command format: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(2);
        case '3' :
                printf("Bad option format: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(3);
        case '4' :
                printf("Last active command is still active: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(4);
        case '5' :
                printf("Command failed: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(5);
        case '8' :
                printf("Bad machine code: %c %c %c \n",
                buff[0],buff[1],buff[2]);
                return(8);
        case '9' :
                buff[0] = buff[2];
                read(video,buff+1,8);
                if (*n1 < 1 || *n1 > 5)
                   {printf("illegal first argument \n"); return(*n1);}
                else if (buff[*n1] == '0') return(-2);
                return(-1);

        default:
                printf("returning undefined code : %c\n",buff[1]);
                return(7);
        }
}


/* environment write */
int v_EW(n1, n2, n3, n4, n5)
int *n1, *n2, *n3, *n4, *n5;
{
        int v_retid();
        buff[0] = 'E';
        buff[1] = 'W';
        buff[2] = '/';
        buff[3] = 'A';
        buff[4] = '1';
        buff[5] = (*n1 == 0) ? '0' : '1';
        buff[6] = 'A';
        buff[7] = '2';
        buff[8] = (*n2 == 0) ? '0' : '1';
        buff[9] = 'C';
        buff[10] = (*n3 == 0) ? '0' : '1';
        buff[11] = 'E';
        buff[12] = (*n4 == 0) ? '0' : '1';
        buff[13] = 'V';
        buff[14] = (*n5 == 0) ? '0' : '1';
        buff[15] = RETURN;
/*      lseek(video,0L,2); */
```

```
        write(video,buff,16);
        return(v_retid());
}


/* v_v_setc */
int v_v_setc(n1, n2)
int *n1, *n2;
{
        int v_retid();
        buff[0] = 'E';
        buff[1] = 'M';
        buff[2] = '/';
        switch (*n1){
        case 1:
                buff[3] = 'A';
                buff[4] = '1';
                break;
        case 2:
                buff[3] = 'A';
                buff[4] = '2';
                break;
        case 3:
                buff[3] = 'C';
                buff[4] = ' ';
                break;
        case 4:
                buff[3] = 'E';
                buff[4] = ' ';
                break;
        case 5:
                buff[3] = 'V';
                buff[4] = ' ';
                break;}

        buff[5] = (*n2 == 0) ? '0' : '1';
        buff[6] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,7);
        return(v_retid());
}


/*  last command */
int v_L()
{
        buff[0] = 'L';
        buff[1] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,2);
        read(video,buff,3);
        if (buff[1] = '9') {
                read(v'deo,buff,1);
                return(buff[2]-'0');
        }
        else return(buff[1]-'0');
}


/*  mode computer */
```

```
int v_MC()
{
        int v_retid();
        buff[0] = 'M';
        buff[1] = 'C';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
        return(v_retid());
}

/*  mode terminal */
int v_MT()
{
        int v_retid();
        buff[0] = 'M';
        buff[1] = 'T';
        buff[2] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,3);
        return(v_retid());
}

/* current frame number */
int v_N()
{
        int v_retid();
        buff[0] = 'N';
        buff[1] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,2);
        return(v_retid());
}

/* quit */
int v_Q()
{
        int v_retid();
        buff[0] = 'Q';
        buff[1] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,2);
        return(v_retid());
}

/* Z command  */
int v_Z()
{
        int v_retid();
        buff[0] = 'Z';
        buff[1] = RETURN;
/*      lseek(video,0L,2); */
        write(video,buff,2);
        return(v_retid());
}

/* v_ptchs command  */
v_ptchs(sum, carry)
int *sum,
char *carry,
```

```
{
    /* if at all return is there, it would only be at the end */
    if (carry[*num - 1] == '@')
        carry[*num - 1] = RETURN;
/*    lseek (video, 0L, 2); */
    write (video, carry, *num);
}

/* v_gtchs command  */
v_gtchs(num, carry)
int *num;
char *carry;
{
    int     i;
    char    c,
           *pt;
    pt = carry;
    if (*num == 0)
        while (read (video, &c, 1) == 1)
            if (c != '\n')
                *pt++ = c;
            else
                return (0);
    else
        for (i = 0; i < *num; read (video, pt++, 1), i++);
}

/* v_flush command  */
v_flush()
{
    char c;
    while(read(video,&c,1) == 1);
}
```

```
;;; -*- Mode: LISP; Syntax: common-lisp; Package: USER; Base: 10; -*-

;                ;;;;;  tapefnslm.lisp  ;;;;;
; This is the tape driver for the LAMBDA LISP machine, started by loading
; this package. This package is COMMON LISP compatible.

(defvar *UZERO* #x30)
(defvar *UONE*  #x31)
(defvar *UTWO*  #x32)
(defvar *UTHRE* #x33)
(defvar *UFOUR* #x34)
(defvar *UFIVE* #x35)
(defvar *USIX*  #x36)
(defvar *USEVE* #x37)
(defvar *UEIGH* #x38)
(defvar *UNINE* #x39)

(defvar *UA* #x41)
(defvar *UB* #x42)
(defvar *UC* #x43)
(defvar *UD* #x44)
(defvar *UE* #x45)
(defvar *UF* #x46)
(defvar *UG* #x47)
(defvar *UH* #x48)
(defvar *UI* #x49)
(defvar *UJ* #x4A)
(defvar *UK* #x4B)
(defvar *UL* #x4C)
(defvar *UM* #x4D)
(defvar *UN* #x4E)
(defvar *UO* #x4F)
(defvar *UP* #x50)
(defvar *UQ* #x51)
(defvar *UR* #x52)
(defvar *US* #x53)
(defvar *UT* #x54)
(defvar *UU* #x55)
(defvar *UV* #x56)
(defvar *UW* #x57)
(defvar *UX* #x58)
(defvar *UY* #x59)
(defvar *UZ* #x5A)

(defvar *UNULL*  #x00)
(defvar *RETURN* #x25)
(defvar *ENTER*  #x25)
(defvar *STAR*   #x2A)
(defvar *COMMA*  #x2C)
(defvar *SLASH*  #x2F)
(defvar *COLON*  #x3A)
(defvar *ATSGN*  #x40)
(defvar *video*)
(defun steal-port-b ()
  (dolist (dev si:all-shared-devices)
    (cond ((string-equal "SDU-SERIAL-B:"
                         (send dev :name))
           (send dev :allocate t)
           (rplaca (send dev :lock) nil)
           (return dev)))))
```

```
(defun vplaydemolmi()
  (with-open-file (*video* "sdu-serial-b:")
    (send *video* :set-baud-rate 9600)
    (loop (print (eval (read))))))

(defun vplaydemosym()
  (with-open-stream (*video* (si:make-serial-stream
                               :unit 2
                               :baud 9600
                               :check-over-run-errors t
                               :ascii-characters t
                               :force-output t))
    (loop (msg (N 1) (eval (read))))))

;;;(defun mygetchar(&optional (n 1))
;;;    (let ((junk (ZL:explodeu (read-char))))
;;;      (cond ((equal junk '(14l))
;;;             (setq junk (ZL:explodeu (read-char)))))
;;;      (do ((ans junk (append ans (ZL:explodeu (read-char)))))
;;;          (tmp 1 (1+ tmp)))
;;;          ((= tmp n) ans))))

(defmacro  vp-getchar(&optional (n 1))
  `(do ((ans nil (cons (send *video* :tyi) ans))
        (tmp 0 (+ 1 tmp)))
       ((= tmp ,n) (nreverse ans))))

;;;(defun vp-putchar(lis)                ; write all the chars in the lis
;;;    (msg (ZL:maknam lis) (N 1)))

(defun myputhchar(lis)
  (dotimes (x lis) (send *video* :tyo x)))

(defun vp-reset()
    "; p"
     ;; flush all the buffers and reset the port so the user can reset
     ;; if in case some thing goes wrong.
)


;;;  * to get the return id and print the err msg if any */
(defun vp-retid(&optional (nl 8))
    (let* ((all (vp-getchar 3))
           (second (cadr all)))
          (cond ((or (= second *STAR*) (= second *UZERO*)) 0)
                ((= second *UONE*)
                 (msg 1 " Illegal command for this machine  " all) 1)
                ((= second *UTWO*)
                 (msg 1 " Bad command format   " all) 2)
                ((= second *UTHRE*)
                 (msg 1 " Bad option format   " all) 3)
                ((= second *UFOUR*)
                 (msg 1 " Last active command is still active   " all) 4)
                ((= second *UFIVE*)
                 (msg 1 " Command failed   " all) 5)
                ((= second *UEIGH*)
                 (msg 1 " Bad machine code   " all) 8)
                ((and (= second *UNINE*) (= nl 8))
                 (setf all (cons (caddr all) (vp-getchar 4)))
```

```
                (vp-getchar 4) , three zeros and a carriage return char
                (c_to_i all))
               ((and (= second *UNINE*) (= n1 1))
                (vp-getchar)
                (let ((jnk (caddr all)))
                     (cond ((= jnk *UZERO*)
                            (msg " No active command ") 0)
                           ((= jnk *UONE*) (msg " vp-ff ") 1)
                           ((= jnk *UTWO*) (msg " vp-g  ") 2)
                           ((= jnk *UTHRE*) (msg " vp-r  ") 3)
                           ((= jnk *UFOUR*) (msg " vp-i  ") 4)
                           ((= jnk *UFIVE*) (msg " vp-pf ") 5)
                           ((= jnk *USIX*) (msg " vp-he ") 6)
                           ((= jnk *USEVE*) (msg " vp-hd ") 7)
                           ((= jnk *UEIGH*) (msg " vp-sf ") 8)
                           ((= jnk *UNINE*) (msg " vp-fr ") 9))))
               (t (msg 1 " Returning undefined code " all) 7))))

... * to change char to integer */
(defun c_to_i(lis)
    (do ((ans 0 (+ (* ans 10) (- (car tmp) *UZERO*)))
         (tmp lis (cdr tmp)))
        ((null tmp) ans)))


... * to send a number */
(defmacro vp-sendnum(x)
    `(vp-putchar (exploden ,x)))

... /* fast forward */
(defmacro vp-fast-forward()
    `(progn ()
            (vp-putchar (list *UF* *UF* *RETURN*))
            (vp-retid)))

... /* fast reverse */
(defmacro vp-rewind()
    `(progn()
           (vp-putchar (list *UF* *UR* *RETURN*))
           (vp-retid)))

... /* go to a frame number n1, & n2 is on off flag */
(defun vp-go-to-frame(&optional (n1 0) (n2 1))
    (vp-putchar (cond ((> n2 0) (list *UG* *SLASH* *UE* *UONE* *COLON*))
                      (t (list *UG* *SLASH* *UE* *UZERO* *COLON*))))
    (vp-sendnum n1)
    (vp-putchar (list *ENTER*))
    (vp-retid))

... /* hold disable */
(defmacro vp-hold-disable()
    `(progn()
           (vp-putchar (list *UH* *UD* *RETURN*))
           (vp-retid)))

... /* hold enable */
(defmacro vp-hold-enable()
    `(progn()
           (vp-putchar (list *UH* *UE* *RETURN*))
           (vp-retid)))
```

```
... /* initialize */
(defmacro vp-initialize()
    `(progn()
           (vp-putchar (list *UI* *RETURN*))
           (vp-retid)))


... /* play forward
...   n1 is frame number to start, n2 is frame number to end
...   n1 = 0 starts from the current position, (default)
...      < 0 start from begining,
...   n2 = 0 stops at the end (default),
...      > 0 stops at that frame number  */
(defun vp-play-forward(&optional (n1 0) (n2 0))
    (vp-putchar (list *UP* *UF*))
    (cond ((= n1 0) (vp-putchar (list *ENTER*)))
          ((< n1 0) (vp-putchar (list *COLON* *UZERO*))
           (cond ((cond ((= n2 0) (vp-putchar (list *ENTER*)))
                        (t (vp-putchar (list *COMMA*))
                           (vp-sendnum n2)
                           (vp-putchar (list *ENTER*)))))))
          (t (vp-putchar (list *COLON*))
             (vp-sendnum n1)
             (cond ((= n2 0) (vp-putchar (list *ENTER*)))
                   (t (vp-putchar (list *COMMA*))
                      (vp-sendnum n2)
                      (vp-putchar (list *ENTER*))))))
    (vp-retid))

... * reject */
(defmacro vp-eject-tape()
    `(progn()
           (vp-putchar (list *UR* *RETURN*))
           (vp-retid)))

... * single step forward a frame */
(defmacro vp-single-frame()
    `(progn()
           (vp-putchar (list *US* *UF* *RETURN*))
           (vp-retid)))

... * home /M -420 */
(defmacro vp-home()
    `(let ((temp))
          (vp-putchar (list *ATSGN* *ENTER*))
          (setf temp (vp-getchar))
          (cond ((equal (car temp) *STAR*) 0)
                (t (msg (N 1) " Returning Undefined code " temp) 7))))

... * environment read */
... * argument n1 can be video, audio1, audio2, videoend, continuous */
(defun vp-environment-read(&optional (n nil))
    (vp-putchar (list *UE* *UR* *RETURN*))
    (let* ((all (vp-getchar 3))
           (second (cadr all)))
          (cond ((or (= second *STAR*) (= second *UZERO*)) 0)
                ((= second *UONE*)
                 (msg 1 " Illegal command for this machine " all) 1)
                ((= second *UTWO*)
```

```
                 (msg 1 " Bad command format : " all) 2)
                ((= second *UTHRE*)
                 (msg 1 " Bad option format : " all) 3)
                ((= second *UFOUR*)
                 (msg 1 " Last active command is still active : " all) 4)
                ((= second *UFIVE*)
                 (msg 1 " Command failed : " all) 5)
                ((= second *UEIGH*)
                 (msg 1 " Bad machine code : " all) 8)
                ((= second *UNINE*)
                 (setf all (cddr (vp-getchar 7)))
                 (vp-getchar)    ; return char
                 (let ((temp (do ((ans nil (cons (cond ((= (car tmp) *UZERO*)
                                                        0)
                                                       (t 1))
                                                 ans))
                                  (tmp all (cdr tmp)))
                                 ((null tmp) (nreverse ans)))))
                      (cond ((null nl) temp)
                            (t (let ((templ (mapname nl)))
                                    (cond ((= templ 0) 7)
                                          (t (nth (- templ 1) temp))))))))
                (t (msg 1 " Returning undefined code : " all) 7))))

(defun mapname(xl)
    (cond ((equal xl 'video) 1)
          ((equal xl 'audiol) 2)
          ((equal xl 'audio2) 3)
          ((equal xl 'videoend) 4)
          ((equal xl 'continuous) 5)
          (t (msg "Illegal first argument : " xl) 0)))

;;; /* environment write */
(defun vp-environment-write(nl n2 n3 n4 n5)
    (vp-putchar (list *UE* *UW* *SLASH* *UV*))
    (vp-putchar (if (= nl 0) (list *UZERO*) (list *UONE*)))
    (vp-putchar (if (= n2 0) (list *UA* *UONE* *UZERO*)
                (list *UA* *UONE* *UONE*)))
    (vp-putchar (if (= n3 0) (list *UA* *UTWO* *UZERO*)
                (list *UA* *UTWO* *UONE*)))
    (vp-putchar (if (= n4 0) (list *UE* *UZERO*) (list *UE* *UONE*)))
    (vp-putchar (if (= n5 0) (list *UC* *UZERO*) (list *UC* *UONE*)))
    (vp-retid))

;;; /* vp-set */
(defun vp-set(nl &optional (n2 1))
    (cond ((member nl '(video audiol audio2 videoend continuous))
           (vp-putchar (list *UE* *UW* *SLASH*))
           (vp-putchar (cond ((equal nl 'video) (list *UV*))
                             ((equal nl 'audiol) (list *UA* *UONE*))
                             ((equal nl 'audio2) (list *UA* *UTWO*))
                             ((equal nl 'vedioend) (list *UE*))
                             ((equal nl 'continuous) (list *UC*))))
           (vp-putchar (list (cond ((or (= n2 0) (= n2 'off)) *UZERO*)
                                   (t *UONE*)) *ENTER*))
           (vp-retid))
          (t (msg (N 1) " Illegal first argument : " nl))))
```

```
;;; /*  last command */
(defmacro vp-last-command()
    `(progn()
           (vp-putchar (list *UL* *ENTER*))
           (vp-retid 1)))


;;; /*  mode computer */
(defmacro vp-mode-computer()
    `(progn()
           (vp-putchar (list *UM* *UC* *ENTER*))
           (vp-retid)))

;;; /*  mode terminal */
(defmacro vp-mode-terminal()
    `(progn()
           (vp-putchar (list *UM* *UT* *ENTER*))
           (vp-retid)))

;;; /* current frame number */
(defmacro vp-frame-number()
    `(progn()
           (vp-putchar (list *UN* *ENTER*))
           (vp-retid)))

;;; /* quit */
(defmacro vp-quit()
    `(progn()
           (vp-putchar (list *UQ* *ENTER*))
           (vp-retid)))

;;; /* Z command  */
(defmacro vp-last-command-status()
    `(progn()
           (vp-putchar (list *UZ* *ENTER*))
           (vp-retid)))


; ----------------
(defun mapname(xl)
    (cond ((equal xl 'video) 1)
          ((equal xl 'audiol) 2)
          ((equal xl 'audio2) 3)
          ((equal xl 'videoend) 4)
          ((equal xl 'continuous) 5)
          (t (msg "Illegal first argument to vp-setc/getc ") 0)))
; --------------------------------------------------------------

(defun vp-help()
    (msg (N 1) "*** The following functions are available ***"))
```

VIDEO DISK PLAYER

Video Disc

Naveen Hota

JAYCOR

Usage

The video disc can be operated from lisp. The routines are written in C. These routines should be loaded into lisp before using them. To use them do the following:

set the EXT switch to 'on' on the backside of the video player. switch on the video player.

On VAX

```
(load 'discfns)
(vd-open)
--- whatever ---
(vd-close)
```

On LMI

```
(load 'discfnslm)
(with-open-file (*video* "sdu:serial-b:")
    (send *video* ':set-baud-rate 1200)
--- whatever ---
)
```

Some times this won't work due to the serial-port being opened by someone and forgot to close. In that case use the following:

```
(with-open-file (*video* (steal-port-b)
    (send *video* ':set-baud-rate 1200)
    ---- whatever ---- )
```

On Symbolics

```
(load 'discfnslm)
(with-open-stream (*video* (si:make-serial-stream ':unit 2
':baud 1200 ':check-over-run-errors t ':ascii-characters t
':force-output t))
  ----- whatever -----)
```

Commands

Following is a description of each of these functions that can be called from lisp. All the commands return an integer code. A returned value of 0 always indicates success. For other values, it prints an error message and returns an integer between 1 and 9.

Name: **vd-open**

Arguments: none

Description: vd-open enables device communication (only on VAX).

Returns 0.

Name: **vd-close**

Arguments: none

Description: vd-close disables the device communication (only on VAX).
Returns 0.

Extended Commands

Extended commands are a combination of the basic commands used often. They are:

Name: **vd-search-frame**

Arguments: n1

Description: Searches for the frame n1 and stills there.
n1 should be 0 <= n1 =< 54000.
Returns 0.

Name: **vd-search-segment**

Arguments: n1

Description: Searches for the beginning of segment n1 and stills there.
n1 - should be 0 <= n1 <= 63
Returns 0.

Name: **vd-search-frame-repeat**

Arguments: n1 n2 n3 n4

Description: Searches for frame n1 and repeats n3 times, playing frames n1 to n2 at speeds specified by n4.

n1 - beginning frame number
n2 - ending frame number
n3 - number of times to repeat (lm:default 1)
n4 = 0 - normal speed (lm:defualt)
= 1 - fast
= -1 - slow

if n1 > n2 then direction is backward,
else direction is forward.

n1 and n2 should be with in 0 and 54000.
n3 should be with in 0 and 15.
Returns 0.

Name: **vd-search-segment-repeat**

Arguments: n1 n2 n3 n4

Description: Searches for segment n1 and repeats n3 times, playing segment n1 to n2 at speeds specified by n4.

n1 - beginning segment number
n2 - ending segment number
n3 - number of times to repeat (lm:defualt 1)
n4 = 0 - normal speed (lm:default)
= 1 - fast
= -1 - slow

if n1 > n2 then direction is backward,
else direction is forward.

n1 and n2 should be with in 0 and 63.
n3 should be with in 0 and 15.
Returns 0.

Name: **vd-search-frmseg-repeat**

Arguments: n1 n2 n3 n4

Description: Searches for frame n1 and repeats n3 times, playing n1 to segment n2 at speeds specified by n4.

n1 - beginning frame number
n2 - ending segment number
n3 - number of times to repeat (lm:default 1)
n4 = 0 - normal speed (lm:default)
= 1 - fast
= -1 - slow

n1 should be with in 0 and 54000.
n2 should be with in 0 and 63.
n3 should be with in 0 and 15.
Returns 0.

Name: **vd-segsave**

Arguments: n1 n2 n3

Description: Sets the segment number n1 to the frames n2 tp n3.

n1 - segment number to be set.
n2 - beginning frame number
n3 - ending frame number.
Returns 0.

Basic Commands

Basic commands are lower-level commands. The user can program using these basic commands according to his needs. Each individual command returns 0 to indicate success. The user has to check the returned code of all the basic commands being used in the user program.

Name: **vd-sendnum**

Arguments: arg1

Description: arg1 should be an integer and should be with in some limits imposed by the context.
While referring to a frame number - 0 <= arg1 <= 54000.
While referring to a segment - 0 <= arg1 <= 63.
While referring to repetition factor - 0 <= arg1 <= 15.
Returns 0.

Name: **vd-play**

Arguments: n1

Description: Plays in the specified direction until the next command.
Direction is indicated by n1.
n1 = 0 => backward, and n1 = 1 => forward. (lm:default)
Returns 0.
Stops at the end or beginning of the disc.

Name: **vd-fast**

Arguments: n1

Description: Fast forwards/backward the disc.
n1 = 0 => fast backward.
n1 = 1 => fast forward (lm:default).
Returns 0.
Stops at the end or beginning of the disc.

Name: **vd-slow**

Arguments: n1

Description: vd-slow plays slowly in the f/b direction.
The speed is 1/3 of the normal play speed.
n1 = 0 => play backwards slowly.
n1 = 1 => play forward slowly (lm:default).
Returns 0.
Stops at the end/beginning of the disc.

Name: **vd-step**

Arguments: n1

Description: vd-fstep steps one frame at a time in the f/b direction.
n1 = 0 => step backward.
n1 = 1 => step forward (lm:default).

Returns 0.
Stops at the end or beginning.

Name: **vd-scan**

Arguments: n1

Description: vd-fscan scans in the f/b direction at a speed of 30 times the normal play speed.
n1 = 0 => scan backward.
n1 = 1 => scan forward (lm:default).
Returns 0.
Stops at the end or beginning.

Name: **vd-stop**

Arguments: none

Description: vd-stop stops the active command that is executing.
Retuns 0.

Name: **vd-enter**

Arguments: none

Description: vd-enter is used to separate individual commands.
Returns 0.

Name: **vd-ce**

Arguments: none

Description: vd-ce is used to cancel the last input while giving commands.
Returns 0.

Name: **vd-menu**

Arguments: none

Description: Gives a menu on the screen.
The disc should have menu choice recorded in it. (None of the current discs have this.)
Returns 0.

Bugs: Tries to find the menu and may not stop finding.
At times it stops at some frame number.
Use vd-cl to cancel this command, if it seems to be not stopping.

Name: **vd-search**

Arguments: none

Description: This is used to search a frame. Won't work as an individual command. Use vd-search-frame. This is used in combination with other commands to search a frame or a segment.
example:

```
(vd-search)     ; search mode
(vd-sendnum n1) ; frame number to be searched
```

(vd-enter) ; indicate the completion of the command
Finds frame number n1, and returns 0.

Name: **vd-repeat**

Arguments: none

Description: This is used in combination with other commands to repeat some frames or segments. Won't work individually.
example:
(vd-search-frame 1000) ; go to frame number 1000
(vd-repeat) ; repeat mode
(vd-sendnum 2000) ; till 2000th frame
(vd-enter)
(vd-sendnum 2) ; repeat 2 times
(vd-enter)
plays the frames 1000 to 2000, 2 times.
Returns 0.

Name: **vd-segment**

Arguments: none

Description: Used in combination with other commands.
Example: (vd-segment) ; segment mode
(vd-sendnum 2) ; segment number
(vd-enter)
(vd-sendnum 200) ; beginning frame number
(vd-enter)
(vd-sendnum 300) ; ending frame number
(vd-enter)
Memorizes segment 2 to be from frames 200 to 300.
Returns 0.

Name: **vd-ch1**

Arguments: n1

Description: vd-ch1 changes the status of audio channel 1.
Depending on the value of n1, it toggles/on/off the status of audio channel 1.
n1 $<$ 0 -- toggles (lm:default)
n1 $=$ 0 -- turns off (regardless of previous status)
n1 $>$ 0 -- turns on (regarless of previous status)
Returns 0.

Name: **vd-ch2**

Arguments: n1

Description: vd-ch2 changes the status of audio channel 2.
Depending on the value of n1, it toggles/on/off the status of audio channel 1.

nl < 0 -- toggles (lm:default)
nl = 0 -- turns off (regardless of previous status)
nl > 0 -- turns on (regarless of previous status)
Returns 0.

Name: **vd-still**

Arguments: none

Description: Freezes at the current frame.
Returns 0.

Name: **vd-index**

Arguments: nl

Description: The index when turned on, displays the current command, status etc in a small rectangle in the upper left corner of the screen.
nl = -1 => toggles index (lm:default).
nl = 0 => turns off index.
nl = 1 => turns on index.
Returns 0.

Name: **vd-dumpin, vd-dumpout**

Arguments: add1

Description: add1 is the address of a block containing 1024 bytes.
These are used to send/receive a block of 1024 bytes.
Won't send any return code in acknowledgement.
Have to send 1025 bytes and get error code as an acknowledgement (returned code).
Returns error code.

Name: **vd-cl**

Arguments: none

Description: vd-cl clears the previous command. This is used when the previous command has errors in it.
example: (vd-search-frame 1000)
(vd-search-frame 2000)
The second command won't be executed if enough time is not lapsed between the first and second search. To clear the second command use vd-cl.

Name: **vd-pgm, vd-pgmend, vd-run, vd-end**

Arguments: none

Description: Used to start, run and end a program.
Can do the programming in lisp.

Name: **vd-memory**

Arguments: none

Description: Memorizes the current frame, which can be called later by vd-msearch.
Returns 0.
example: (vd-play 1)
say we are at 1202th frame.
(vd-memory) memorizes this frame.
& the play continues and stops at the end.
Now (vd-msearch) would take back to 1202th frame.
vd-memory can remember only one number.
Always the current number replaces the previous number.

Name: **vd-msearch**

Arguments: none

Description: Used to find the previously memorized frame.
Returns 0.

Name: **vd-skip**

Arguments: none

Description: Used to skip a frame. Usually used after vd-still to look at the next frame.
Returns 0.

Name: **vd-int**

Arguments: none

Description: Initializes the player, puts the head at the beginning of the disc and turns on audio channels 1&2.
Returns 0.

Name: **vd-review**

Arguments: none

Description: Its a control instruction. Returns 0.

Name: **vd-mode**

Arguments: n1

Description: Depending on the value of n1 it sets the mode.
n1 = -1 toggles between frame mode and segment mode (lm:default)
n1 = 0 sets to frame mode.
n1 = 1 sets to segment mode.
Returns 0.

Name: **vd-continue**

Arguments: none

Description: Used to continue the previous operation after a vd-still.
Returns 0.

Name: **vd-motor**

Arguments: nl

Description: Turns the motor on/off
nl = 0 => off
nl = 1 => on (lm:default)
Returns 0.

```
;
;               ***** discfns.l *****
;
; This set of LISP calls reads in and binds LISP function names to
; the C code which actually does the work.  Run once at startup time.
;

(cfasl "/usr/local/lib/video/vdisc.o" "_v_open" "vd-open" "integer-function")

(getaddress "_v_close" "vd-close" "integer-function")
(getaddress "_SetTerm" "vd-set" "integer-function")
(getaddress "_RstTerm" "vd-reset" "integer-function")
(getaddress "_v_retid" "vd-retid" "integer-function")
(getaddress "_v_sendnum" "vd-sendnum" "integer-function")

(getaddress "_v_zero" "vd-zero" "integer-function")
(getaddress "_v_play" "vd-play" "integer-function")
(getaddress "_v_fast" "vd-fast" "integer-function")
(getaddress "_v_slow" "vd-slow" "integer-function")
(getaddress "_v_step" "vd-step" "integer-function")
(getaddress "_v_scan" "vd-scan" "integer-function")
(getaddress "_v_still" "vd-still" "integer-function")
(getaddress "_v_stop" "vd-stop" "integer-function")
(getaddress "_v_enter" "vd-enter" "integer-function")
(getaddress "_v_ce" "vd-ce" "integer-function")
(getaddress "_v_menu" "vd-menu" "integer-function")
(getaddress "_v_search" "vd-search" "integer-function")
(getaddress "_v_repeat" "vd-repeat" "integer-function")
(getaddress "_v_segment" "vd-segment" "integer-function")
(getaddress "_v_ch1" "vd-ch1" "integer-function")
(getaddress "_v_ch2" "vd-ch2" "integer-function")
(getaddress "_v_index" "vd-index" "integer-function")
(getaddress "_v_dumpin" "vd-dumpin" "integer-function")
(getaddress "_v_dumpout" "vd-dumpout" "integer-function")
(getaddress "_v_cl" "vd-cl" "integer-function")
(getaddress "_v_pgm" "vd-pgm" "integer-function")
(getaddress "_v_run" "vd-run" "integer-function")
(getaddress "_v_end" "vd-end" "integer-function")
(getaddress "_v_memory" "vd-memory" "integer-function")
(getaddress "_v_msearch" "vd-msearch" "integer-function")
(getaddress "_v_skip" "vd-skip" "integer-function")
(getaddress "_v_int" "vd-int" "integer-function")
(getaddress "_v_review" "vd-review" "integer-function")
(getaddress "_v_mode" "vd-mode" "integer-function")
(getaddress "_v_continue" "vd-continue" "integer-function")
(getaddress "_v_motor" "vd-motor" "integer-function")
(getaddress "_v_sleep" "vd-sleep" "integer-function")

(getaddress "_v_search_frame" "vd-search-frame" "integer-function")
(getaddress "_v_search_segment" "vd-search-segment" "integer-function")
(getaddress "_v_search_frame_repeat" "vd-search-frame-repeat
    "integer-function")
(getaddress "_v_search_segment_repeat" "vd-search-segment-repeat
    "integer-function")
(getaddress "_v_search_frmseg_repeat" "vd-search-frmseg-repeat
    "integer-function")
(getaddress "_v_segsave" "vd-segsave" "integer-function")
```

```
/*
        vdisc.c

        This set of C functions is loaded in by the lisp code in "discfns.l"
        and does the acttual work of the LISP functions. Compile with:

                cc -O -c vdisc.c

*/

#include <sgtty.h>
#define UNULL 0x00
#define UCOMP 0x01
#define UERRO 0x02
#define UPGME 0x04
#define UNTAR 0x05
#define UACK  0x0A
#define UNACK 0x0B
#define UZERO 0x30
#define UONE  0x31
#define UTWO  0x32
#define UTHRE 0x33
#define UFOUR 0x34
#define UFIVE 0x35
#define USIX  0x36
#define USEVE 0x37
#define UEIGH 0x38
#define UNINE 0x39
#define UFPLA 0x3A
#define UFFAS 0x3B
#define UFSLO 0x3C
#define UFSTE 0x3D
#define UFSCA 0x3E
#define USTOP 0x3F
#define UENTE 0x40
#define UCE   0x41
#define UMENU 0x42
#define USEAR 0x43
#define UREPE 0x44
#define USEGM 0x45
#define UCH1N 0x46
#define UCH1F 0x47
#define UCH2N 0x48
#define UCH2F 0x49
#define URPLA 0x4A
#define URFAS 0x4B
#define URSLO 0x4C
#define URSTE 0x4D
#define URSCA 0x4E
#define USTIL 0x4F
#define UINDN 0x50
#define UINDF 0x51
#define UDUMN 0x52
#define UDUMT 0x53
#define USEMO 0x54
#define UFRAM 0x55
#define UCL   0x56
#define UPGM  0x57
#define URUN  0x58
```

```
#define UEND  0x59
#define UMEMO 0x5A
#define UMSEA 0x5B
#define USKIP 0x5C
#define UINT  0x5D
#define UREVI 0x5E
#define UMODE 0x5F
#define UADDI 0x60
#define UCONT 0x61
#define UMOTN 0x62
#define UMOTF 0x63
#define UCH1  0x64
#define UCH2  0x65
#define UINDE 0x66
#define USTAI 0x67
#define UDISI 0x68

static int video,
char *tbuf,

v_open()
{
        int SetTerm(),
        if((video = open("/dev/ttyi0", 2)) < 0) return (-1),
        SetTerm(),
        return(0),
}

v_close()
{
        int RstTerm(),
        RstTerm(),
        close(video),
        return(0),
}

struct sgttyb oldmodes,newmodes,

SetTerm()
{
        ioctl(video, TIOCGETP, &oldmodes),
        ioctl(video, TIOCGETP, &newmodes),
        newmodes.sg_flags  = RAW,
        newmodes.sg_flags &= ~ECHO,
        newmodes.sg_ispeed = B1200,
        newmodes.sg_ospeed = B1200,
        ioctl(video, TIOCSETN, &newmodes),
}

RstTerm()
{
        ioctl(video, TIOCSETN, &oldmodes),
}

int v_retid()
{
        int v_ci(),
        read (video,tbuf,1),
        switch (*tbuf){
        case '0'
```

```
        case UNULL:
        case UACK:
        case UCOMP:
        case UPGME:
                return(0);
        case UNTAR:
                printf("number not found\n");
                v_cl();
                return(1);
        case UERRO:
                printf("invalid command\n");
                v_cl();
                return(2);
        case UNACK:
                printf("command not in valid range\n");
                v_cl();
                return(3);
        default:
                printf("returning undefined code - %x\n",*tbuf);
                v_cl();
                return(5);
        }
}

int mputchar(x)
char x;
{
        int v_retid();
        write(video,&x,1);
        return(v_retid());
}

int v_sendnum(x)
int *x;
{
        int     bi = 14;
        int     y,temp,v_retid();
        char    buff[15];
        y = *x;
        if (y == 0) {
            buff[0] = '0';
            write(video,buff,1);
            return (v_retid());
        }
        while (y > 0 && bi >= 0) {
                buff[bi--] = y % 10 + '0';
                y /= 10;
        }
        while (bi < 14){
                write(video, buff+bi+1, 1);
                bi++;
                if ((temp = v_retid()) > 0) return(temp);
        }
        return(temp);
}

int v_zero()
{
        int mputchar();
        return(mputchar(UZERO));
}
```

```
}

int v_play(nl)
int *nl;
{
        int mputchar();
        return(mputchar((*nl > 0) ? UFPLA : URPLA));
}


int v_fast(nl)
int *nl;
{
        int mputchar();
        return(mputchar((*nl > 0) ? UFFAS : URFAS));
}


int v_slow(nl)
int *nl;
{
        int mputchar();
        return(mputchar((*nl > 0) ? UFSLO : URSLO));
}


int v_step(nl)
int *nl;
{
        int mputchar();
        return(mputchar((*nl > 0) ? UFSTE : URSTE));
}


int v_scan(nl)
int *nl;
{
        int mputchar();
        return(mputchar((*nl > 0) ? UFSCA : URSCA));
}


int v_still()
{
        int mputchar();
        return(mputchar(USTIL));
}


int v_stop()
{
        int mputchar();
        return(mputchar(USTOP));
}
int v_enter()
{
        int rz,mputchar();
        rz = mputchar(UENTE);
        sleep(1);
```

```
        return(rz);
}

int v_ce()
{
        int mputchar();
        return(mputchar(UCE));
}

int v_menu()
{
        int mputchar();
        return(mputchar(UMENU));
}

int v_search()
{
        int mputchar();
        return(mputchar(USEAR));
}

int v_repeat()
{
        int mputchar();
        return(mputchar(UREPE));
}

int v_segment()
{
        int mputchar();
        return(mputchar(USEGM));
}


int v_ch1(nl)
int *nl;
{
        int mputchar();
        return(mputchar((*nl > 0) ? UCH1N :
        (*nl == 0) ? UCH1F : UCH1));
}


int v_ch2(nl)
int *nl;
{
        int mputchar();
        return(mputchar((*nl > 0) ? UCH2N :
        (*nl == 0) ? UCH2F : UCH2));
}


int v_index(nl)
int *nl;
{
        int mputchar();
        return(mputchar((*nl > 0) ? UINDN :
        (*nl == 0) ? UINDF : UINDE));
}
```

```
int v_dumpin()
{
        int mputchar();
        return(mputchar(UDUMN));
}

int v_dumpout()
{
        int mputchar();
        return(mputchar(UDUMT));
}


int v_cl()
{
        int mputchar();
        return(mputchar(UCL));
}

int v_pgm()
{
        int mputchar();
        return(mputchar(UPGM));
}

int v_run()
{
        int mputchar();
        return(mputchar(URUN));
}

int v_end()
{
        int mputchar();
        return(mputchar(UEND));
}

int v_memory()
{
        int mputchar();
        return(mputchar(UMEMO));
}

int v_msearch()
{
        int mputchar();
        return(mputchar(UMSEA));
}

int v_skip()
{
        int mputchar();
        return(mputchar(USKIP));
}

int v_int()
{
        int mputchar();
        return(mputchar(UINT));
}
```

```
int v_review()
{
        int mputchar(),
        return(mputchar(UREVI)),
}

int v_mode(nl)
int *nl,
{
        int mputchar(),
        return(mputchar((*nl < 0) ? UMODE :
        (*nl == 0) ? UFRAM : USEMO)),
}

int v_continue()
{
        int mputchar(),
        return(mputchar(UCONT)),
}

int v_motor(nl)
int *nl,
{
        int mputchar(),
        return(mputchar((*nl > 0) ? UMOTN : UMOTF)),
}

int v_sleep(nl)
int *nl,
{
        *nl = (*nl < 0) ? - *nl : *nl,
        sleep(*nl),
        return(0),
}

int v_search_frame(num)
int *num,
{
        int rz,mputchar(), v_sendnum(),
        if (*num < 0 || *num > 54000) return(4),
        if ((rz = mputchar(USEAR)) > 0) return(rz),
        if ((rz = v_sendnum(num)) > 0) return(rz),
        rz = mputchar(UENTE),
        sleep(2),
        return(rz),
}

int v_search_segment(num)
int *num,
{
        int rz,mputchar(), v_sendnum(),
        if (*num < 0 || *num > 63) return(4),
        if ((rz = mputchar(USEAR)) > 0) return(rz),
        if ((rz = mputchar(UMODE)) > 0) return(rz),
        if ((rz = v_sendnum(num)) > 0) return(rz),
        rz = mputchar(UENTE),
        sleep(2),
        return(rz),
}
```

```
int v_search_frame_repeat(bnum,endnum,ntmes,speed)
int *bnum, *endnum, *ntmes, *speed,
{
        int sign, rz,mputchar(),v_sendnum(),
        char speedc,
        if (*bnum < 0 || *bnum > 54000 || *endnum < 0 || *endnum > 54000)
                return(4),
        if ((sign = v_search_frame(bnum)) > 0) return(sign),
        sign = (*bnum > *endnum) ? 0 : 1,
        *ntmes = (*ntmes < 0 || *ntmes > 15) ? 15 : *ntmes,

        if ((rz = mputchar(UREPE)) > 0) return(rz),
        if ((rz = v_sendnum(endnum)) > 0) return(rz),
        if (*speed == -1 &&
            ((rz = mputchar((sign == 0) ? URSLO : UFSLO)) > 0)) return(rz),
        if (*speed == 1 &&
            ((rz = mputchar((sign == 0) ? URFAS : UFFAS)) > 0)) return(rz),
        if ((rz = mputchar(UENTE)) > 0) return(rz),
        if ((rz = v_sendnum(ntmes)) > 0) return(rz),
        return(mputchar(UENTE)),
}

int v_search_segment_repeat(snum1,snum2,ntmes,speed)
int *snum1, *snum2, *ntmes, *speed,
{
        int rz,sign,mputchar(),v_sendnum(),
        if (*snum1 < 0 || *snum1 > 63 ||
            *snum2 < 0 || *snum2 > 63) return(4),
        if ((rz = v_search_segment(snum1)) > 0) return(rz),
        *ntmes = (*ntmes < 0 || *ntmes > 15) ? 15 : *ntmes,
        sign = (*snum1 > *snum2) ? 0 : 1,

        if ((rz = mputchar(UREPE)) > 0) return(rz),
        if ((rz = v_sendnum(snum2)) > 0) return(rz),
        if (*speed == -1 &&
            ((rz = mputchar((sign == 0) ? URSLO : UFSLO)) > 0)) return(rz),
        if (*speed == 1 &&
            ((rz = mputchar((sign == 0) ? URFAS : UFFAS)) > 0)) return(rz),
        if ((rz = mputchar(UENTE)) > 0) return(rz),
        if ((rz = v_sendnum(ntmes)) > 0) return(rz),
        return(mputchar(UENTE)),
}

int v_search_frmseq_repeat(fnum,snum,ntmes,speed)
int *fnum, *snum, *ntmes, *speed,
{
        int rz,mputchar(),v_sendnum(),
        char speedc,
        if (*snum < 0 || *snum > 63 || *fnum < 0 || *fnum > 54000) return(4),
        if ((rz = v_search_frame(fnum)) > 0) return(rz),
        *ntmes = (*ntmes < 0 || *ntmes > 15) ? 15 : *ntmes,

        if ((rz = mputchar(UREPE)) > 0) return(rz),
        if ((rz = mputchar(UMODE)) > 0) return(rz),
        if ((rz = v_sendnum(snum)) > 0) return(rz),
        if (*speed == -1 &&
            (rz = mputchar(UFSLO)) > 0) return(rz),
        else if (*speed == 1 &&
            (rz = mputchar(UFFAS)) > 0)  return(rz),
```

```
        if ((rz = mputchar(UENTE)) > 0) return(rz);
        if ((rz = v_sendnum(ntmes)) > 0) return(rz);
        return(mputchar(UENTE));
}

int v_seqsave(n1, n2, n3)
int *n1, *n2, *n3;
{
        int rz,v_sendnum(),mputchar();
        if (*n1 < 0 || *n1 > 63)
        {
                printf(" Illegal segment number : %d\n",*n1);
                return(4);
        };
        if (*n2 > 54000 || *n3 > 54000)
        {
                printf(" Illegal frame number > 54000 .\n");
                return(4);
        };
        if ((rz = mputchar(USEGM)) > 0) return(rz);
        if ((rz = v_sendnum(n1)) > 0) return(rz);
        if ((rz = mputchar(UENTE)) > 0) return(rz);
        if (*n2 >= 0 && (rz = v_sendnum(n2)) > 0) return(rz);
        if ((rz = mputchar(UENTE)) > 0) return(rz);
        if (*n3 >= 0 && (rz = v_sendnum(n3)) > 0) return(rz);
        return(mputchar(UENTE));
}
```

```
;;; -*- Mode: LISP; Syntax: common-lisp; Package: USER; Base: 10; -*-

;
;                ;;;;;  discfnslm.lisp  ;;;;;
;
; This is the package of functions which interfaces tto the disk player
; from the LAMBDA machine. It is COMMON LISP compatible.

(defvar *UNULL* #x00)
(defvar *UCOMP* #x01)
(defvar *UERRO* #x02)
(defvar *UPGME* #x04)
(defvar *UNTAR* #x05)
(defvar *UACK*  #x0A)
(defvar *UNACK* #x0B)
(defvar *UZERO* #x30)
(defvar *UONE*  #x31)
(defvar *UTWO*  #x32)
(defvar *UTHRE* #x33)
(defvar *UFOUR* #x34)
(defvar *UFIVE* #x35)
(defvar *USIX*  #x36)
(defvar *USEVE* #x37)
(defvar *UEIGH* #x38)
(defvar *UNINE* #x39)
(defvar *UFPLA* #x3A)
(defvar *UFFAS* #x3B)
(defvar *UFSLO* #x3C)
(defvar *UFSTE* #x3D)
(defvar *UFSCA* #x3E)
(defvar *USTOP* #x3F)
(defvar *UENTE* #x40)
(defvar *UCE*   #x41)
(defvar *UMENU* #x42)
(defvar *USEAR* #x43)
(defvar *UREPE* #x44)
(defvar *USEGM* #x45)
(defvar *UCH1N* #x46)
(defvar *UCH1F* #x47)
(defvar *UCH2N* #x48)
(defvar *UCH2F* #x49)
(defvar *URPLA* #x4A)
(defvar *URFAS* #x4B)
(defvar *URSLO* #x4C)
(defvar *URSTE* #x4D)
(defvar *URSCA* #x4E)
(defvar *USTIL* #x4F)
(defvar *UINDN* #x50)
(defvar *UINDF* #x51)
(defvar *UDUMN* #x52)
(defvar *UDUMT* #x53)
(defvar *USEMO* #x54)
(defvar *UFRAM* #x55)
(defvar *UCL*   #x56)
(defvar *UPGM*  #x57)
(defvar *URUN*  #x58)
(defvar *UEND*  #x59)
(defvar *UMEMO* #x5A)
(defvar *UMSEA* #x5B)
(defvar *USKIP* #x5C)
```

```
(defvar *UINT*  #x5D)
(defvar *UREVI* #x5E)
(defvar *UMODE* #x5F)
(defvar *UADDI* #x60)
(defvar *UCONT* #x61)
(defvar *UMOTN* #x62)
(defvar *UMOTF* #x63)
(defvar *UCH1*  #x64)
(defvar *UCH2*  #x65)
(defvar *UINDE* #x66)
(defvar *USTAI* #x67)
(defvar *UDISI* #x68)

(defvar *video*)
(defun steal-port-b ()
  (dolist (dev si:all-shared-devices)
    (cond ((string-equal "SDU-SERIAL-B:"
                         (send dev :name))
       (send dev :allocate t)
       (rplaca (send dev :lock) nil)
       (return dev)))))

(defun vdiscdemolmi()
  (with-open-file  (*video* "sdu-serial-b:")
    (send *video* ':set-baud-rate 9600)
    (loop (print (eval (read))))))

(defun vdiscdemosym()
  (with-open-stream (*video* (si:make-serial-stream
                              ':unit 2
                              ':baud 9600
                              ':check-over-run-errors t
                              ':ascii-characters t
                              ':force-output t))
    (loop (msg (N 1) (eval (read))))))

;;;(defun mygetchar()
;;;    (car (ZL:exploden (read-char))))
(defmacro  mygetchar()
  `(send *video* ':tyi))
;;;(defun myputchar(atm &optional (n 0));sleep time before reading again
;;;    ;;;"J.P"
;;;    ;;; returns vd-retid
;;;    (msg (ZL:ascii atm) (N 1))
;;;    (or (= n 0) (vd-sleep n))
;;;    (vd-retid))
(defun myputchar(chr &optional (n 0));sleep time before reading in again
  (send *video* ':tyo chr)
  (or (= n 0) (sleep n))
  (vd-retid))


(defun vd-reset()
    ;;;"J.P"
    ;;; flush all the buffers and reset the port so the user can reset
    ;;; if in case some thing goes wrong.
)


;;;(defun mygetchar() "J.P")
```

```
;;;(defun myputchar(ch)
;;;     "J.P"
;;;     ... returns vd-retid)

(defun vd-retid()
    (let ((retch (mygetchar)))
         (cond ((or (= retch *UZERO*)
                    (= retch *UNULL*)
                    (= retch *UACK*)
                    (= retch *UCOMP*)
                    (= retch *UPGME*)) 0)
               ((= retch *UNTAR*) (msg 1 "Number not found ") 1)
               ((= retch *UERRO*) (msg 1 "Invalid Command  ") 2)
               ((= retch *UNACK*)
                (msg 1 "Command not in valid range ") 3)
               (t (msg 1 "Returning undefiend code " retch) 5))))

(defun vd-sendnum(x)
    (do ((ans 0 (myputchar (car tmpl)))
         (tmpl (ZL:exploden x) (cdr tmpl)))
        ((or (> ans 0) (null tmpl)) ans)))

(defmacro vd-zero() `(myputchar *UZERO*))

(defmacro vd-play(&optional (nl 1))
    `(myputchar (cond ((> ,nl 0) *UPPLA*)
                      (t *URPLA*))))

(defmacro vd-fast(&optional (nl 1))
    `(myputchar (cond ((> ,nl 0) *UFFAS*)
                      (t *URFAS*))))

(defmacro vd-slow(&optional (nl 1))
    `(myputchar (cond ((> ,nl 0) *UFSLO*)
                      (t *URSLO*))))

(defmacro vd-step(&optional (nl 1))
    `(myputchar (cond ((> ,nl 0) *UFSTE*)
                      (t *URSTE*))))

(defmacro vd-scan(&optional (nl 1))
    `(myputchar (cond ((> ,nl 0) *UFSCA*)
                      (t *URSCA*))))

(defmacro vd-still()
    `(myputchar *USTIL*))

(defmacro vd-stop()
    `(myputchar *USTOP*))

(defmacro  vd-enter(&optional (sltime 1))
   `(myputchar *UENTE* ,sltime))


(defmacro vd-ce()
    '(myputchar *UCE*))

(defmacro vd-menu()
    `(myputchar *UMENU*))
```

```
(defmacro vd-search()
    `(myputchar *USEAR*))

(defmacro vd-repeat()
    `(myputchar *UREPE*))

(defmacro vd-segment()
    `(myputchar *USEGM*))

(defmacro vd-ch1(&optional (nl -1))
    `(myputchar (cond ((> ,nl 0) *UCH1N*)
                      ((= ,nl 0) *UCH1F*)
                      (t *UCH1*))))

(defmacro vd-ch2(&optional (nl -1))
    `(myputchar (cond ((> ,nl 0) *UCH2N*)
                      ((= ,nl 0) *UCH2F*)
                      (t *UCH2*))))

(defmacro vd-index(&optional (nl -1))
    `(myputchar (cond ((> ,nl 0) *UINDN*)
                      ((= ,nl 0) *UINDF*)
                      (t *UINDE*))))

(defmacro vd-dumpin()
    `(myputchar *UDUMN*))

(defmacro vd-dumpout()
    `(myputchar *UDUMT*))

(defmacro vd-cl()
    `(myputchar *UCL*))

(defmacro vd-pgm()
    `(myputchar *UPGM*))

(defmacro vd-run()
    `(myputchar *URUN*))

(defmacro vd-end()
    `(myputchar *UEND*))

(defmacro vd-memory()
    `(myputchar *UMEMO*))

(defmacro vd-msearch()
    `(myputchar *UMSEA*))

(defmacro vd-skip()
    `(myputchar *USKIP*))

(defmacro vd-int()
    `(myputchar *UINT*))

(defmacro vd-review()
    `(myputchar *UREVI*))

(defmacro vd-mode(&optional (nl -1))
    `(myputchar (cond ((< ,nl 0) *UMODE*)
                      ((= ,nl 0) *UPRAM*)
```

```
                    (t *USEMO*))))

(defmacro vd-continue()
    `(myputchar *UCONT*))

(defmacro vd-motor(&optional (nl 1))
    `(myputchar (cond ((> ,nl 0) *UMOTN*)
                    (t *UMOTF*))))

(defmacro vd-sleep(&optional (nl 1))
    `(cond ((< ,nl 0) (sleep (minus ,nl)) 0)
           (t (zleep ,nl) 0)))  ; lisp sleep function ******


(defun vd-search-frame(num)
    (prog (tmp)
          (if (or (< num 0) (> num 54000)) (return 4))
          (if (> (setf tmp (myputchar *USEAR*)) 0) (return tmp))
          (if (> (setf tmp (vd-sendnum num)) 0) (return tmp))
          (return (vd-enter 2))))

(defun vd-search-segment(num)
    (prog (tmp)
          (if (or (< num 0) (> num 63)) (return 4))
          (if (> (setf tmp (myputchar *USEAR*)) 0) (return tmp))
          (if (> (setf tmp (myputchar *UMODE*)) 0) (return tmp))
          (if (> (setf tmp (vd-sendnum num)) 0) (return tmp))
          (return (vd-enter 2))))

(defun vd-search-frame-repeat(bnum  enum &optional (ntmes 1) (speed 0))
    (prog (tmp)
          (if (< bnum 0) (setf bnum 0))
          (if (> bnum 54000) (setf bnum 54000))
          (if (< enum 0) (setf enum 0))
          (if (> enum 54000) (setf enum 54000))
          (if (> (setf tmp (vd-search-frame bnum)) 0) (return tmp))
          (if (< ntmes 0) (setf ntmes 0))
          (if (> ntmes 15) (setf ntmes 15))
          (if (> (setf tmp (myputchar *UREPE*)) 0) (return tmp))
          (if (> (setf tmp (vd-sendnum enum)) 0) (return tmp))
          (cond ((< speed 0)
                 (if (> (setf tmp (myputchar (if (> bnum enum)
                                                 *URSLO* *UFSLO*))) 0)
                     (return tmp)))
                ((> speed 0)
                 (if (> (setf tmp (myputchar (if (> bnum enum)
                                                 *URFAS* *UFFAS*))) 0)
                     (return tmp))))
          (if (> (setf tmp (myputchar *UENTE*)) 0) (return tmp))
          (if (> (setf tmp (vd-sendnum ntmes)) 0) (return tmp))
          (return (vd-enter 2))))

(defun vd-search-segment-repeat(bnum enum
                                &optional (ntmes 1) (speed 0))
    (prog (tmp)
          (if (< bnum 0) (setf bnum 0))
          (if (> bnum 63) (setf bnum 63))
          (if (< enum 0) (setf enum 0))
          (if (> enum 63) (setf enum 63))
          (if (> (setf tmp (vd-search-segment bnum)) 0) (return tmp))
```

```
          (if (< ntmes 0) (setf ntmes 0))
          (if (> ntmes 15) (setf ntmes 15))
          (if (> (setf tmp (myputchar *UREPE*)) 0) (return tmp))
          (if (> (setf tmp (vd-sendnum enum)) 0) (return tmp))
          (cond ((< speed 0)
                 (if (> (setf tmp (myputchar (if (> bnum enum)
                                                 *URSLO* *UFSLO*))) 0)
                     (return tmp)))
                ((> speed 0)
                 (if (> (setf tmp (myputchar (if (> bnum enum)
                                                 *URFAS* *UFFAS*))) 0)
                     (return tmp))))
          (if (> (setf tmp (myputchar *UENTE*)) 0) (return tmp))
          (if (> (setf tmp (vd-sendnum ntmes)) 0) (return tmp))
          (return (vd-enter 2))))

(defun vd-search-frmseg-repeat (fnum snum &optional (ntmes 1) (speed 0))
    (prog (tmp)
          (if (< fnum 0) (setf fnum 0))
          (if (> fnum 63) (setf fnum 63))
          (if (< snum 0) (setf snum 0))
          (if (> snum 54000) (setf snum 63))
          (if (> (setf tmp (vd-search-frame fnum)) 0) (return tmp))
          (if (< ntmes 0) (setf ntmes 0))
          (if (> ntmes 15) (setf ntmes 15))
          (if (> (setf tmp (myputchar *UREPE*)) 0) (return tmp))
          (if (> (setf tmp (myputchar *UMODE*)) 0) (return tmp))
          (if (> (setf tmp (vd-sendnum snum)) 0) (return tmp))
          (cond ((< speed 0)
                 (if (> (setf tmp (myputchar *UFSLO*)) 0)
                     (return tmp)))
                ((> speed 0)
                 (if (> (setf tmp (myputchar *UFFAS*)) 0)
                     (return tmp))))
          (if (> (setf tmp (myputchar *UENTE*)) 0) (return tmp))
          (if (> (setf tmp (vd-sendnum ntmes)) 0) (return tmp))
          (return (vd-enter 2))))

(defun vd-segsave(snum  bnum enum)
    (prog (tmp)
          (cond ((or (< snum  0) (> snum 63))
                 (msg 1 "Illegal segment number  " snum)
                 (msg 1 "   should be with in 0 and 63 ")
                 (return 4)))
          (if (< bnum 0) (setf bnum 0))
          (if (< enum 0) (setf enum 0))
          (if (> bnum 0) (setf bnum 54000))
          (if (> enum 0) (setf enum 54000))
          (if (> bnum enum) (psetf bnum enum enum bnum))

          (if (> (setf tmp (myputchar *USEGM*)) > 0) (return tmp))
          (if (> (setf tmp (vd-sendnum snum)) > 0) (return tmp))
          (if (> (setf tmp (myputchar *UENTE*)) > 0) (return tmp))
          (if (> (setf tmp (vd-sendnum bnum)) > 0) (return tmp))
          (if (> (setf tmp (myputchar *UENTE*)) > 0) (return tmp))
          (if (> (setf tmp (vd-sendnum enum)) > 0) (return tmp))
          (return (vd-enter 2))))

(defun vd-help()
    (mac (N 1) "  mygetchar")
```

```
(msg (N 1) "  myputchar")
(msg (N 1) "  vd-ce")
(msg (N 1) "  vd-ch1")
(msg (N 1) "  vd-ch2")
(msg (N 1) "  vd-cl")
(msg (N 1) "  vd-close")
(msg (N 1) "  vd-continue")
(msg (N 1) "  vd-dumpin")
(msg (N 1) "  vd-dumpout")
(msg (N 1) "  vd-end")
(msg (N 1) "  vd-enter")
(msg (N 1) "  vd-fast")
(msg (N 1) "  vd-index")
(msg (N 1) "  vd-int")
(msg (N 1) "  vd-memory")
(msg (N 1) "  vd-menu")
(msg (N 1) "  vd-mode")
(msg (N 1) "  vd-motor")
(msg (N 1) "  vd-msearch")
(msg (N 1) "  vd-open")
(msg (N 1) "  vd-pgm")
(msg (N 1) "  vd-play")
(msg (N 1) "  vd-repeat")
(msg (N 1) "  vd-reset")
(msg (N 1) "  vd-retid")
(msg (N 1) "  vd-review")
(msg (N 1) "  vd-run")
(msg (N 1) "  vd-scan")
(msg (N 1) "  vd-search")
(msg (N 1) "  vd-search-frame")
(msg (N 1) "  vd-search-frame-repeat")
(msg (N 1) "  vd-search-frmseg-repeat "  )
(msg (N 1) "  vd-search-segment")
(msg (N 1) "  vd-search-segment-repeat")
(msg (N 1) "  vd-segment")
(msg (N 1) "  vd-segsave")
(msg (N 1) "  vd-sendnum")
(msg (N 1) "  vd-skip")
(msg (N 1) "  vd-sleep")
(msg (N 1) "  vd-slow")
(msg (N 1) "  vd-step")
(msg (N 1) "  vd-still")
(msg (N 1) "  vd-stop")
(msg (N 1) "  vd-zero"))
```

LA50 PRINTER

```
;;; -*- Mode: LISP; Syntax: Zetalisp; Package: USER; Lowercase: Yes; Base: 10;

;               =====  print.lisp  =====
;
; This package allows the user to print screen dumps to the la50 printer
; connected to SDU serial port B. Any rectangular portion of the screen
; may be dumped. The printer is not the fastest in the world...

(defun steal-port-b()
  (dolist (dev si:all-shared-devices)
    (cond ((string-equal "SDU-SERIAL-B"
                         (send dev :name))
           (send dev :allocate t)
           (rplaca (send dev :lock) nil)
           (return dev)))))


(defun get-num (b1 &optional (b2 #b0) (b3 #b0) (b4 #b0) (b5 #b0) (b6 #b0))
  (+ (* 1 b1) (* 2 b2) (* 4 b3) (* 8 b4) (* 16 b5) (* 32 b6) 63))

(defun l1p50pr(arhd wd ht)
  (let* ((nwd (* 6 (ceiling ht 6)))
         (nht wd)
         (nwar-t (make-array (list nht nwd)
                             :type 'art-1b :initial-value #b0))
         tch nwd1 nht1 nht2
         )

    (copy-array-contents arhd nwar-t)
    (with-open-file (lp50 "sdu-serial-b:" :direction :io)
      (send lp50 :set-baud-rate 4800)
      (send lp50 ':tyo #o033)                    ;; escape
      (send lp50 ':tyo #o120)                    ;; P
      (send lp50 ':tyo #o161)                    ;; q
      (send lp50 ':tyo #o014)                    ;; form feed
      (global:dotimes (i (ceiling nht 612))      ;; for each strip
        (setf nht1 (* i 612))
        (global:dotimes (j (ceiling nwd 6))      ;; for each colump of 6pixels
          (setf nwd1 (* j 6))
          ;; for each row of <= 712 pixels
          (global:dotimes (k (min 612 (- wd nht1)))
            (setq nht2 (+ nht1 k)
                  tch (get-num (aref nwar-t nwd1 nht2)
                               (aref nwar-t (+ 1 nwd1) nht2)
                               (aref nwar-t (+ 2 nwd1) nht2)
                               (aref nwar-t (+ 3 nwd1) nht2)
                               (aref nwar-t (+ 4 nwd1) nht2)
                               (aref nwar-t (+ 5 nwd1) nht2)))
            (send lp50 ':tyo tch)
            (send lp50 ':tyo tch));; twice because of the aspect ratio 1:2
          (send lp50 ':tyo #o055)                ;; next line
          (and (zerop (mod nwd1 131))
               (send lp50 ':tyo #o014)))
        (send lp50 ':tyo #o014)))))

(comment
(defun l11p50pr(arhd wd ht)
  (let ((htl (* 6 (ceiling ht 6)))
        nwar-t
        tch nwd1 nht1 nwd2)
```

```
    (cond ((equal ht htl) (setf nwar-t arhd))
          (t (setf nwar-t (make-array (list wd htl) :type 'art-1b))
             (copy-array-contents arhd nwar-t)))

    (with-open-file (lp50 "sdu-serial-b:" :direction :io)
      (send lp50 :set-baud-rate 4800)
      (send lp50 :set-output-buffer-size 1023)
      (send lp50 :set-force-output-p nil)
      (send lp50 ':tyo #o033)                    ;; escape
      (send lp50 ':tyo #o120)                    ;; P
      (send lp50 ':tyo #o161)                    ;; q
      (send lp50 ':tyo #o014)                    ;; form feed
      (global:dotimes (i (ceiling wd 500))       ;; for each strip
        (setf nwd1 (* i 500))
        (global:dotimes (j (ceiling htl 6))      ;; for each colump of 6pixels
          (setf nht1 (* j 6))
          ;; for each row of <= 572, pixels 8in.
          (global:dotimes (k (min 500 (- wd nwd1)))
            (setq nwd2 (+ nwd1 k)
                  tch (get-num (aref nwar-t nht1 nwd2)
                               (aref nwar-t (+ 1 nht1) nwd2)
                               (aref nwar-t (+ 2 nht1) nwd2)
                               (aref nwar-t (+ 3 nht1) nwd2)
                               (aref nwar-t (+ 4 nht1) nwd2)
                               (aref nwar-t (+ 5 nht1) nwd2)))
            (send lp50 ':tyo tch)
            (send lp50 ':tyo tch));; twice because of the aspect ratio 1:2
          (send lp50 ':tyo #o055)                ;; next line
          (and (zerop (mod nht1 131))
               (send lp50 ':tyo #o014)))         ; form feed (next page)
        (send lp50 ' tyo #o014)))))
)
```

```
/*
        crtk.c

        This set of functions allows the user to interface to the RAMTEK
        displays in a uniform manner. All fucntions are used directly in
        the user's C programs by includinge this file at the top of the user's
        source with:

                #include crtk.c

        or by initially compiling with:

                cc -O -c crtk.c

        and including it on the compile call line is:

                cc yourprog.c crtk.o

*/

static char buff[4200];
static int ramtek;

r_open(orientation)
int orientation;
{
        if((ramtek = open("/dev/rtk", 2)) < 0) return(-1);
        buff[0] = orientation & 0x01;
        buff[1] = 0x27;
        if(write(ramtek, buff, 2) != 2) return(-1);
        return(0);
}

r_close()
{
        close(ramtek);
        return(0);
}

r_line(x0, y0, x1, y1, color)
int x0, y0, x1, y1, color;
{
        buff[0] = 0x03; buff[1] = 0x0E;
        buff[2] = 0x02; buff[3] = 0x80;
        buff[4] = color & 0xFF; buff[5] = 0x00;
        buff[6] = x0 & 0xFF; buff[7] = (x0 >> 8) & 0x07;
        buff[8] = y0 & 0xFF; buff[9] = (y0 >> 8) & 0x03;
        buff[10] = 0x04; buff[11] = 0x00;
        buff[12] = x1 & 0xFF; buff[13] = (x1 >> 8) & 0x07;
        buff[14] = y1 & 0xFF; buff[15] = (y1 >> 8) & 0x03;

        if(write(ramtek, buff, 16) != 16) return (-1);
        return (0);
}

r_rect(x0, y0, x1, y1, color)
int x0, y0, x1, y1, color;
{
        buff[0] = 0x02; buff[1] = 0x09;
```

```
        buff[2] = 0x44; buff[3] = 0x00;
        buff[4] = color & 0xFF; buff[5] = 0x00;
        buff[6] = x0 & 0xFF; buff[7] = (x0 >> 8) & 0x07;
        buff[8] = y0 & 0xFF; buff[9] = (y0 >> 8) & 0x03;
        buff[10] = x1 & 0xFF; buff[11] = (x1 >> 8) & 0x07;
        buff[12] = y1 & 0xFF; buff[13] = (y1 >> 8) & 0x03;

        if(write(ramtek, buff, 14) != 14) return (-1);
        return (0);
}

r_erase()
{
        return(r_rect(0,0,1279,1023,0));
}

r_reset()
{
        buff[0] = 0x00; buff[1] = 0x05;

        if(write(ramtek, buff, 2) != 2) return (-1);
        return (0);
}

r_text(x0, y0, color, textptr)
int x0, y0, color;
char *textptr;
{
        return(rtext(x0, y0, color, 0, textptr));
}

r_text2(x0, y0, fcolor, bcolor, textptr)
int x0, y0, fcolor, bcolor;
char *textptr;
{
        return(rtext(x0, y0, fcolor, bcolor, textptr));
}

rtext(x0, y0, fcolor, bcolor, textptr)
int x0, y0, fcolor, bcolor;
char *textptr;
{
        register int wsize, length;
        register char *bptr;

        buff[0] = 0x0B; buff[1] = 0x0C;
        buff[2] = 0x46; buff[3] = 0x80;
        buff[4] = fcolor & 0xFF; buff[5] = 0x00;
        buff[6] = bcolor & 0xFF; buff[7] = 0x00;
        buff[8] = 0x00; buff[9] = 0x00;
        buff[10] = 0x00; buff[11] = 0x00;
        buff[12] = 0xFF; buff[13] = 0x04;
        buff[14] = 0xFF; buff[15] = 0x03;
        buff[16] = x0 & 0xFF; buff[17] = (x0 >> 8) & 0x07;
        buff[18] = y0 & 0xFF; buff[19] = (y0 >> 8) & 0x03;

        length = strlen(textptr);

        if(length <= 0) return(-1);
        if(length > 78) length = 78;  /* Maximum size string which fits buff */
```

```
        buff[20] = length & 0xFF, buff[21] = 0x00,
        if((wsize = 22 + length) & 1) wsize++,    /* number of bytes for write */

        bptr = &buff[22],
        while(length--) *bptr++ = *textptr++,

        if(write(ramtek, buff, wsize) < wsize) return(-1),
        return (0),
}

r_wtab(values, load_table, start, nvals, select_table)
char *values,
int load_table, start, nvals, select_table,
{
        register i, j, k,

        j = 0,
        if(nvals) {       /* Then must load before selecting */

        /* Load Auxiliary Memory command -- Device 0 */

                buff[0] = 0x00,
                buff[1] = 0x03,

        /* Table start address -- 16bit start, not entry start */

                buff[2] = (i = start * 2),
                buff[3] = ((i >> 8) + (load_table * 2)) & 0xFF,

         * Number of bytes to load -- 4 times number of entries */

                buff[4] = (i = nvals * 4) & 0xFF,
                buff[5] = (i >> 8) & 0xFF,

                j = 6,
                for(i=0,i<nvals,i++) {
                        buff[j++] = *values++,    /* Blue */
                        buff[j++] = *values++,    /* Green */
                        buff[j++] = *values++,    /* Red */
                        buff[j++] = *values++,    /* And any blink... */
                }
        }

        k = j,          /* Either zero or (6 + (nvals * 4)) */

        buff[k++] = 0x00,       /* Select table    */
        buff[k++] = 0x03,

        buff[k++] = 0,          /* Table number */
        buff[k++] = (select_table * 2) & 0x06,

        buff[k++] = 0,          /* Just selecting so no actual writing... */
        buff[k++] = 0,

        if(write(ramtek, buff, k) != k) return(-1),
        return(select_table),
}

r_wimage(imagedata, xstart, ystart, numpts, scan_style, operation)
```

```
char *imagedata,
int xstart, ystart, numpts, scan_style, operation,
{
        register i, j, rnumpts,

        i = 0,
        buff[i++] = 0x0F,               /* Flags (RP, OF1, OF2, DF) */
        buff[i++] = 0x0A,               /* Write Image instruction */

        buff[i++] = 0x80,               /* OF1 -- Scan, ... */
        buff[i++] = 0x88,               /* ...COP, Function */

        buff[i++] = 0x20,               /* OF2 -- Image Mode... */
        buff[i++] = 0x00,               /* ...Nothing */

        buff[i++] = scan_style & 0x07,          /* Scan direction */
        buff[i++] = 0x00,

        buff[i++] = operation & 0x0F,           /* Function */
        buff[i++] = 0x00,

        buff[i++] = xstart & 0xFF,              /* COP at first */
        buff[i++] = (xstart >> 8) & 0x07,
        buff[i++] = ystart & 0xFF,
        buff[i++] = (ystart >> 8) & 0x03,

        buff[i++] = 0x01,                       /* Image Mode */
        buff[i++] = 0x00,

        buff[i++] = (rnumpts = numpts) & 0xFF,
        buff[i++] = (rnumpts >> 8) & 0x07,

        j = 0,
        while(rnumpts--) buff[i++] = imagedata[j++],
        if(write(ramtek,buff,i) != i) return(-1),

        return(0),
}

r_logo(x, y)
int x, y,
{
        r_rect(x, y, x+175, y+70, 4),
        r_text2(x+49, y+10, 3, 4, "NAVY CENTER"),
        r_text2(x+77, y+20, 3, 4, "FOR"),
        r_text2(x+28, y+30, 3, 4, "APPLIED RESEARCH"),
        r_text2(x+77, y+40, 3, 4, "IN"),
        r_text2(x+7, y+50, 3, 4, "ARTIFICIAL INTELLIGENCE"),
}
```

```
/*
        lisprtk.c

        This set of functions is compiled, then loaded into the lisp
        environment with "cfasl", with appropriate binding of LISP
        function names to the actual C routines.
        Compile with:

                cc -O -c lisprtk.c

*/


static char buff[150];
static int ramtek;

r_open()
{
        if((ramtek = open("/dev/rtk", 2)) < 0) return (-1);
        return(0);
}

r_close()
{
        close(ramtek);
        return(0);
}

r_line(x0, y0, x1, y1, color)
int *x0, *y0, *x1, *y1, *color;
{
        buff[0] = 0x03; buff[1] = 0x0E;
        buff[2] = 0x02; buff[3] = 0x80;
        buff[4] = *color & 7; buff[5] = 0x00;
        buff[6] = *x0 & 0xFF; buff[7] = (*x0 >> 8) & 0x01;
        buff[8] = *y0 & 0xFF; buff[9] = (*y0 >> 8) & 0x01;
        buff[10] = 0x04; buff[11] = 0x00;
        buff[12] = *x1 & 0xFF; buff[13] = (*x1 >> 8) & 0x01;
        buff[14] = *y1 & 0xFF; buff[15] = (*y1 >> 8) & 0x01;

        if(write(ramtek, buff, 16) != 16) return (-1);
        return (0);
}

r_rect(x0, y0, x1, y1, color)
int *x0, *y0, *x1, *y1, *color;
{
        buff[0] = 0x02; buff[1] = 0x09;
        buff[2] = 0x44; buff[3] = 0x00;
        buff[4] = *color & 7; buff[5] = 0x00;
        buff[6] = *x0 & 0xFF; buff[7] = (*x0 >> 8) & 0x01;
        buff[8] = *y0 & 0xFF; buff[9] = (*y0 >> 8) & 0x01;
        buff[10] = *x1 & 0xFF; buff[11] = (*x1 >> 8) & 0x01;
        buff[12] = *y1 & 0xFF; buff[13] = (*y1 >> 8) & 0x01;

        if(write(ramtek, buff, 14) != 14) return (-1);
        return (0);
}
```

```
r_erase()
{
        buff[0] = 0x02; buff[1] = 0x09;
        buff[2] = 0x44; buff[3] = 0x00;
        buff[4] = 0x00; buff[5] = 0x00;
        buff[6] = 0x00; buff[7] = 0x00;
        buff[8] = 0x00; buff[9] = 0x00;
        buff[10] = 0xFF; buff[11] = 0x01;
        buff[12] = 0xFF; buff[13] = 0x01;

        if(write(ramtek, buff, 14) != 14) return (-1);
        return (0);
}

r_reset()
{
        buff[0] = 0x00; buff[1] = 0x05;

        if(write(ramtek, buff, 2) != 2) return (-1);
        return (0);
}

r_text(x0, y0, color, textptr)
int *x0, *y0, *color;
char *textptr;
{
        register int wsize, length;
        register char *bptr;

        buff[0] = 0x0B; buff[1] = 0x0C;
        buff[2] = 0x46; buff[3] = 0x80;
        buff[4] = *color & 7; buff[5] = 0x00;
        buff[6] = 0x00; buff[7] = 0x00;
        buff[8] = 0x00; buff[9] = 0x00;
        buff[10] = 0x00; buff[11] = 0x00;
        buff[12] = 0xFF; buff[13] = 0x01;
        buff[14] = 0xFF; buff[15] = 0x01;
        buff[16] = *x0 & 0xFF; buff[17] = (*x0 >> 8) & 0x01;
        buff[18] = *y0 & 0xFF; buff[19] = (*y0 >> 8) & 0x01;

        length = strlen(textptr);

        if(length != 0) return(-1);
        if(length > 78) length = 78;   /* Maximum size string which fits buff */

        buff[20] = length & 0xFF; buff[21] = 0x00;
        if((wsize = 22 + length) & 1) wsize++;   /* number of bytes for write */

        bptr = &buff[22];
        while(length--) *bptr++ = *textptr++;

        if(write(ramtek, buff, wsize) < wsize) return(-1);
        return (0);
}
```

END

4-1-87

DTIC